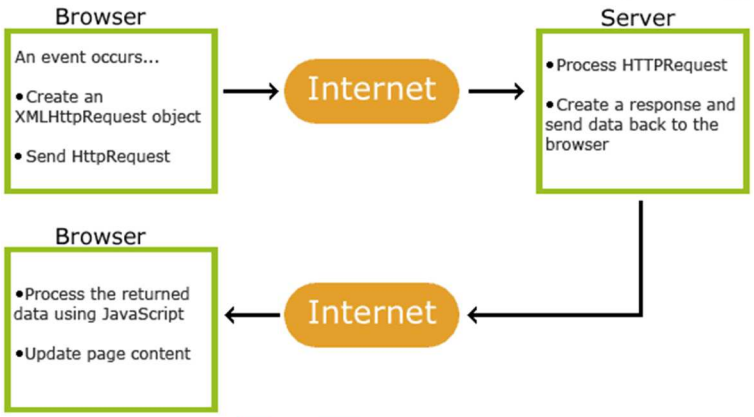
AJAX

AJAX je tehnologija koja koristi HTTP protokol da učita podatke bez ponovnog učitavanja stranice. Naziv AJAX je skraćenica za Asynchronous JavaScript and XML. XML će biti tema kursa *Service Application Development*.

Korišćenjem ove tehnologije možemo da šaljemo podatke ka serveru i primamo odgovor asinhrono. Time se postiže dinamičnost stranice, jer se sadržaj brže učitava u realnom vremenu, bez potrebe za osvežavanjem kompletne stranice.

Na slici 9.1. možemo videti objašnjenje kako AJAX funkcioniše.



Slika 9.1. AJAX – prikaz jednog poziva[1]

Browser – pregledač kroz određene događaje (npr. klikom na dugme) kreira i šalje upit koristeći XMLHttpRequest objekat.

Naš pregledač šalje zahtev preko interneta na server; server preuzima naš zahtev i obrađuje ga. Nakon obrade našeg poziva sačinjava odgovor i šalje ga nazad do našeg pregledača. Ono što se događa na serveru nije vidljivo korisniku.

Pregledač preuzima odgovor i putem JavaScripta ga obrađuje. Tako obrađeni odgovor se koristi kako bi se ažurirao sadržaj naše stranice.

AJAX koristi sledeću kombinaciju:

- XMLHttpRequest objekat, koji je već podržan od strane pregledača i spada u ugrađene objekte JavaScripta (nisu potrebne dodatne instalacije ili biblioteke); XMLHttpRequest objekat ćemo koristiti kasnije u praktičnim primerima kako bismo poslali zahtev ka serveru;
- JavaScript i HTML DOM, kako bismo prikazali vrednosti ili zamenili postojeće.

[1] https://www.w3schools.com/xml/ajax_intro.asp

Pošto smo u prethodnoj lekciji već naučili šta je HTML DOM i kako ga koristimo, preći ćemo odmah na XMLHttpRequest objekat i na mogućnosti koje nam on pruža.

Kao što smo već rekli, XMLHttpRequest je objekat koji već postoji u našem pregledaču i način na koji pravimo instancu tog objekta je:

```
var xhttp = new XMLHttpRequest();
```

Ovakav način je prilagođen za novije pregledače, a za starije pregledače IE5 i IE6 se koristi drugačiji način pravljenja instance:

```
var xhttp = new ActiveXObject("Microsoft.XMLHTTP");
```

U daljem radu i primerima ćemo koristiti standardni, noviji način instanciranja.

U tabeli 9.1. nalaze se metode objekta XMLHttpRequest. Neke od ovih metoda ćemo koristiti u praktičnim primerima.

Metoda	Opis
new XMLHttpRequest()	Kreira instancu XMLHttpRequest objekta
abort()	Zaustavlja sve trenutne zahteve
getAllResponseHeaders()	Vraća sve informacije iz headera
getResponseHeader()	Vraća određene informacije o headeru
open(*method, url, async, user, psw*)	Određuje zahtev *method:* prosleđuje se metoda (npr. GET) *url:* lokacija gde se šalje zahteva *sync:* true (asinhroni) ili false (sinhroni) (opcioni parametar) *user:* korisničko ime (opcioni parametar) *psw:* lozinka (opcioni parametar)
send()	Šalje zahtev ka serveru
setRequestHeader()	Ova metoda se koristi za postavljanje headera. Header se dodaje tako što se postavi ključ–vrednost par prilikom slanja

Tabela 9.1. Metode XMLHttpRequest objekta

U tabeli 9.2. se nalaze sva svojsta XMLHttpRequest objekta. Neke od svojstava ćemo koristiti u praktičnim primerima.

Svojstvo	Opis
onreadystatechange	Definiše funkciju koja će se pozvati kada se readyState svojstvo promeni
readyState	U ovom svojstvu se čuva status XMLHttpRequest 0: zahtev nije inicijalizovan 1: uspostavljena je konekcija sa serverom 2: zahtev je primljen 3: zahtev se obrađuje 4: zahtev je završen i odgovor je spreman
responseText	Vraća odgovor kao string – tekst

responseXML	Vraća odgovor kao XML objekat
status	Vraća status zahtev 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Vraća tekst statusa (npr. "OK" ili "Not Found")

Tabela 9.2. Svojstva XMLHttpRequest objekta

Proći ćemo kroz dva primera upotrebe AJAX-a korišćenjem XMLHttpRequest objekta. Prvi primer šalje zahtev koristeći GET metodu, a drugi koristeći POST metodu.

Implementacije klijenta (JavaScript) i servera (Python) ćemo prikazati na primeru gde server sadrži rečnik imena i prezimena zvan names_dict, a klijent pokušava ili da dobavi prezime za dato ime (GET) ili da doda nove vrednosti tom rečniku (POST) koristeći upitne stringove.

Pitanje

Izabrati metodu XMLHttpRequest objekta:

- onreadystatechange
- readyState
- responseText
- **setRequestHeader()**

Objašnjenje:

Ponuđeni odgovori onreadystatechange, readyState i responseText su svojstva, dok je setRequestHeader() metoda.

Python server

Za server koristimo već poznati kod iz kursa *Python Net Programming*. Jedina izmena jeste dodavanje linije:

```
self.send_header('Access-Control-Allow-Origin', '*')
```

unutar send_response_to_client() funkcije. Ova linija nam omogućava da server kontaktiramo sa bilo kog klijenta.

Ovu skriptu ćemo sačuvati pod imenom http_local_server.py.

Fajl http_local_server.py

```
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import parse_qs
names_dict = {'john':'smith',
              'david':'jones',
              'michael':'johnson',
```

```
                'chris':'lee'}

    class RequestHandler(BaseHTTPRequestHandler):
        def do_GET(self):
            self.log_message("Incoming GET request...")
            try:
                name = parse_qs(self.path[2:])['name'][0]
            except:
                self.send_response_to_client(404, 'Incorrect parameters
    provided')
                self.log_message("Incorrect parameters provided")
                return

            if name in names_dict.keys():
                self.send_response_to_client(200, names_dict[name])
            else:
                self.send_response_to_client(400, 'Name not found')
                self.log_message("Name not found")

        def do_POST(self):
            self.log_message('Incoming POST request...')
            data = parse_qs(self.path[2:])
            try:
                names_dict[data['name'][0]] = data['last_name'][0]
                self.send_response_to_client(200, names_dict)
            except KeyError:
                self.send_response_to_client(404, 'Incorrect parameters
    provided')
                self.log_message("Incorrect parameters provided")

        def send_response_to_client(self, status_code, data):
            # Send OK status
            self.send_response(status_code)
            # Send headers
            self.send_header('Content-type', 'text/plain')
            self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()

            # Send the response
            self.wfile.write(str(data).encode())

    server_address = ('127.0.0.1', 8080)
    http_server = HTTPServer(server_address, RequestHandler)
    http_server.serve_forever()
```

Za pokretanje servera potrebno je pokrenuti komandni prozor i pozicionirati se u direktorijum gde se nalazi fajl http_local_server.py. Potrebno je pokrenuti skriptu komandom python http_local_server.py.

Slanje GET zahtva

U ovom primeru koristićemo događaj onclick koji će biti postavljen na dugme Get i koji će pozivati funkciju loadDoc().

Unutar funkcije, pre samog slanja zahteva ka serveru, potrebno je preuzeti vrednost (ime) koju je korisnik uneo u input polje. To radimo koristeći DOM getElementById metodu sa value atributom, odnosno linijom:

```
var name = document.getElementById("name").value;
```

Zatim kreiramo instancu objekta XMLHttpRequest i koristimo metodu send() kako bi zahtev bio poslat ka serveru.

Pre korišćenja send() i open() metode, neophodno je upotrebiti svojstvo onreadystatechange, koje će omogućiti da nakon odgovora servera izmenimo sadržaj.

Svojstvo onreadystatechange definiše funkciju koja će zameniti tekst unutar p elementa preko DOM getElementById() metode.

Sa servera preuzimamo odgovor linijom this.responseText i ispisujemo je unutar p elementa. Ključnu reč this koristimo da bismo se pozvali na svojstvo tog objekta u instanci koju smo kreirali.

Metodi open() prosleđujemo tri parametra: metodu, koja je u našem slučaju GET, zatim URL koji odgovara našem Python serveru i na kraju vrednost True, koja označava da želimo asinhronu komunikaciju.

Kod prosleđivanja URL-a unutar funkcije open() koristimo konkatenaciju kako bismo unutar upitnog stringa nalepili vrednost koju je korisnik uneo u input polju, koju čuvamo u promenljivoj name.

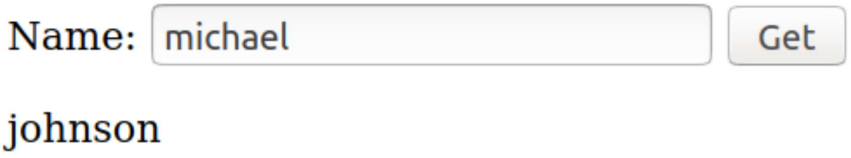
Primer 1

```
<!DOCTYPE html>
<html>
<head>
      <title> AJAX</title>
</head>
<body>
      <label>Name:</label>
      <input type="text" id="name">
      <button onclick="loadDoc()">Get</button>
      <p id="text"></p>
      <script type="text/javascript">
            function loadDoc() {
                  var name = document.getElementById("name").value;
                  var xhttp = new XMLHttpRequest();
                  xhttp.onreadystatechange = function () {
                        document.getElementById("text").innerHTML =
                                    this.responseText;

                  }
                  xhttp.open( "GET",
"http://127.0.0.1:8080/?name="+name, true);
                  xhttp.send();
            }
      </script>
</body>
</html>
```

Na slici 9.2. se vidi prikaz ekrana sa input poljem, dok se na slici 9.3. vidi prikaz ekrana na kome je prikazan i rezultat, odnosno vrednost koju je server vratio kao odgovor na poslati zahtev.

Slika 9.2. Prikaz stranice pre slanja GET zahteva



Slika 9.3. Prikaz stranice nakon slanja GET zahteva

Slanje POST zahteva

Slanje POST zahteva je veomo sličano slanju GET zahteva. U ovom primeru je dodata još jedna labela, sa nazivom Surname, kao i <input> polje gde će se upisivati i prezime koje želimo da pošaljemo serveru pored samog imena.

Server preuzima ime i prezime i dodaje ih u listu, koju potom vraća kao odgovor. I ime i prezime prosleđujemo kroz sam URL.

Primer 1

```
<!DOCTYPE html>
<html>
<head>
	<title> AJAX</title>
</head>
<body>
	<label>Name:</label>
	<input type="text" id="name">
	<br>
	<label>Surname:</label>
	<input type="text" id="surname">
	<br>
	<button onclick="loadDoc()">Get</button>
	<script type="text/javascript">
		function loadDoc() {
			var name = document.getElementById("name").value;
			var surname =
document.getElementById("surname").value;
			var xhttp = new XMLHttpRequest();
			xhttp.onreadystatechange = function () {
				console.log(this.responseText);
			}
```

```
                    xhttp.open( "POST",
    "http://127.0.0.1:8080/?name="+name+
                    "&last_name="+surname, true);
                    xhttp.send();
                }
            </script>
    </body>
    </html>
```

Slika 9.4. Prikaz izgleda stranice pre slanja POST zahteva

Rezutat će biti vraćen u sledećem formatu unutar konzole:

```
{'john': 'smith', 'david': 'jones', 'michael': 'johnson', 'chris': 'lee',
'name': 'surname'}
```

Obeleženi kod, odnosno ključ name i vrednost surname, jesu vrednosti koje je korisnik uneo putem input polja.

Rezime

- AJAX je tehnologija koja koristi HTTP protokol. AJAX nam omogućuje slanje zahteva ka serveru.
- Metode koja AJAX koristi su XMLHttpRequest(), abort(), getAllResponseHeaders(), getResponseHeader(), open(*method, url, async, user, psw*), send(), send(*string*) i setRequestHeader().
- Svojstva koja AJAX koristi su onreadystatechange, readyState, responseText, responseXML, status i statusText.
- Da bi jedan zahtev izveden korišćenjem AJAX tehnologije bio uspešan, potrebno je napraviti instancu XMLHttpRequest objekta, definisati zahtev putem open() metode, poslati zahtev metodom send() i svojstvom onreadystatechange definisati funkciju koja će se pozvati nakon promene readyState.